Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

Virtual and Dynamic Hierarchical Architecture: an overlay network topology for discovering grid services with high performance*

HUANG Li-can (黄理灿)†[1], WU Zhao-hui (吴朝晖)[1], PAN Yun-he (潘云鹤)[2]

([1]*College of Computer Science, Zhejiang University, Hangzhou 310027, China*)

([2]*Ministry's intelligent virtual research center, Hangzhou 310027, China*)

†E-mail: lchuang@cs.zju.edu.cn

Received Jan. 10, 2003; revision accepted June 23, 2003

Abstract: This paper presents an overlay network topology called Virtual and Dynamic Hierarchical Architecture (VDHA) for discovering Grid services with high performance. Service discovery based on VDHA has scalable, autonomous, efficient, reliable and quick responsive. We propose two service discovery algorithms. Full Search Query and Discovery Protocol (FSQDP) discovers the nodes that match the request message from all N nodes, which has time complexity $O(\log N)$, space complexity $O(n_{vg})$ (n_{vg} being node numbers of each virtual group), and message-cost $O(N)$, and Domain-Specific Query and Discovery Protocol (DSQDP) searches nodes in only specific domains with time complexity $O(n_{vg})$, space complexity $O(n_{vg})$, and message-cost $O(n_{vg})$. In this paper, we also describe VDHA, its formal definition, and Grid Group Management Protocol.

Key words: VDHA, Grid, Protocol, Peer-to-peer, Service discovery
Document code: A **CLC number:** TP391.0

INTRODUCTION

Grid (Foster *et al*., 2001) technology is one of the most important ones to appear in recent years. The recent big progress is that scientists (Foster *et al*., 2002; Roure *et al*., 2001; Rana and Walker, 2002) proposed service-oriented architectures such as Open Grid Services Architecture (OGSA) (Foster *et al*., 2002) that integrated the so-called computational/data Grid architecture (Foster *et al*., 2001) with Web services (Grid Web Services Workshop, 2001).

In the service-oriented architecture of Grid, how to find Grid services is an important issue. Globus (Foster and Kesselman, 1997) defines a single, unified access mechanism for a wide range of information, called the Metacomputing Directory Service (MDS) (Foster and Kesselman, 1997). Building on the data representation and application programming interface defined by the Lightweight Directory Access Protocol (LDAP) (Wahl *et al*., 1997), MDS defines a framework in which information of interest in distributed computing applications can be represented. Information is structured as a set of entries, where each entry comprises zero or more attribute-value pairs. In Globus, Grid Information Index Servers (GIISs) (Foster *et al*., 2001) is used to support arbitrary views on resource subsets. Grid Resource Registration Protocol (GRRP)

* Project supported by virtual cooperative research granted by the Chinese Ministry of Education

(Foster *et al*., 2001) is used to register resources. Grid Resource Information Protocol (GRIP) (Foster *et al*., 2001) is used to access information on entities. Web service uses another framework to find the services. It uses WSDL (Christensen *et al*., 2001) to describe services. WSDL is an XML format language for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure oriented information. WSFL (Web services Flow Language) (Web Services Flow Language, 2001) and XLANG (Web Services for Business Process Design, 2001) describe how services can be composed together, and the behavior/interaction protocol of a Web service. Web service framework uses UDDI (Universal Description, Discovery and Integration) (UDDI, 2001) to enable online registry and the publishing and dynamic discovery of the services offered by businesses.

The above services discoveries are centralized, have bad scalability, and a single point of failure.

Iamnitchi and Foster (2001) combine P2P technologies with Grid technologies to discover resources. But algorithms they used are not effective. Although P2P technologies based on Distributed_Hash_table (DHT) technologies such as Pastry (Rowstron and Druschel, 2001), CAN (Ratnasamy *et al*., 2001), and CHORD (Stoica *et al*., 2001) are effective in time complexity, they cannot be used for discovering services which are needed to be described with a lot of entities or semantic languages and which are discovered by partial-match searching.

Chander *et al*.(2002) proposed NEVRLATE for scalable and expressive peer-to-peer (P2P) networking efficient resource discovery. It maintained two two-dimensional sets of servers, for registration to occur in one 'horizontal' dimension, and lookup to occur in the other 'vertical' dimension. It needs to register every service into the square of N (N is the number of all nodes).

Most Internet search-type lookup services fail to be responsive. Search engines like "Google" delay response of dynamic changed services in days.

We present a scalable service discovery based on Virtual and Dynamic Hierarchical Architecture (VDHA) (some ideas were formed in the paper (Huang *et al*., 2002a; 2002b; 2002c) to solve the above problems. Our method is efficient, reliable, and rapid in response and fully searching.

OVERVIEW OF VDHA

Description of VDHA

VDHA is a dynamic and virtual hierarchical architecture (Fig.1) in which Grid nodes are grouped virtually. Nodes can join the group and leave the group dynamically. The groups are virtually hierarchical, with one root-layer, several middle-layers, and many leaf virtual groups (these groups are called VOs). Among these nodes of VOs, one (just one) node (called gateway node) in each group is chosen to form upper-layer groups, whose nodes form upper-upper-layer groups in the same way, and in this way the process is repeated until a root-layer group is formed. In the same group all nodes can be the gateway node located not only in the low-layer group, but also in the upper-layer group. Gateway nodes forward the low- layer group's status information to all the nodes in the upper-layer group, and distribute the upper-layer group's status information to all the nodes in the lower-layer group.

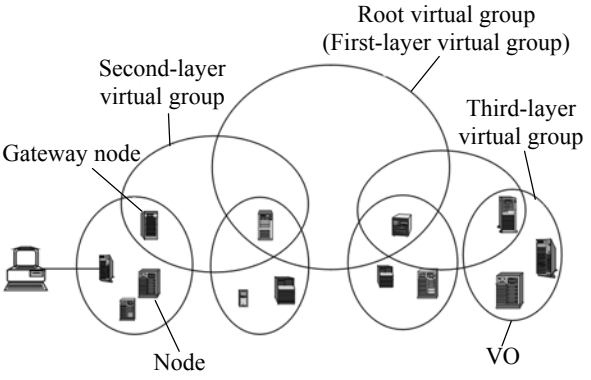

Fig.1 Structure of VDHA

Note: There are 13 nodes in the grid system. These nodes are grouped as 4 VOs. The number of nodes in each VO is 4,3,3,3 respectively. From each VO we choose one node as gateway node to form two upper-layer groups with each having 2 nodes. Then from these two groups, one node each was chosen to form a root group

Generally, the nodes in the same group have the same domain with similar properties. The nodes from within the same group exchange information more frequently than the nodes from within different groups. The numbers of nodes in a VO can be dynamically changed in that the node can dynamically join and leave the VO. A VO may join and leave the Grid system as a whole, and this autonomous property makes the large scalable systems possible.

Formal definition of VDHA

Definition 1 Grid node (denoted by p) is the node in the Grid system. All p form a set PS, that is, $PS=\{p_i \mid i \in N\}$, $N=\{1\ldots n\}$, where, n is the number of the Grid nodes, each p_i has ID (usually Internet IP address).

Definition 2 Entrance node (denoted by ent) is a Grid node, which is an entrance point for users to login to the Grid system.

Definition 3 Client host (denoted by cli) is an apparatus (such as desktop computer, PDA, mobile computer, etc.) used by users to login to the Grid system and to do business. Client host can login to the Grid system via any entrance node.

Definition 4 Gateway node (denoted by gn) is a Grid node with coordinate functions in several different layer virtual groups. Symbol $gn^i_{\alpha_i} \in VG^i_{\alpha_i}$ means that the gateway node is in the ith layer group with name α_i.

Definition 5 Virtual group (denoted by VG) is formed virtually by the Grid nodes. VG^i_α means the group is in the ith layer and the name of this virtual group is α. The virtual group is identified by its group name and layer number.

Definition 6 Coordinator of virtual group (denoted by cvg) is a gateway node taking coordinate functions in the virtual group. The symbol cvg^i_α ($cvg^i_\alpha \in VG^i_\alpha$) means that it is a gateway node in the ith layer α-named virtual group which functions as coordinator.

Definition 7 Virtual group tree (denoted by VGT) is hierarchical tree formed by virtual groups.

In VGT there is a root virtual group (denoted by RVG), many leaf VGs called virtual organization (denoted by VO). $VO^m_{\alpha_m}$ means that the virtual organization is in the mth layer and its name is α_m.

The order of layers is counted from RVG, which is defined as the first-layer VG.

VG except VO is formed purely by gateway nodes. VO is formed by Grid nodes with one (and just one) gateway node.

RVG cannot be a VO, and VO can be within all the layers except the first layer.

N^i_α is the number of nodes in VG^i_α,

N^i_g is the number of virtual groups in the ith-layer of VGT.

Definition 8 VDHA is a virtual group tree with depth of at least two layers. VDHA has dynamic properties in the number of Grid nodes, layers and virtual groups, virtual group compositions, and so on.

In VDHA, we have the following properties:

(1) $VG^i_\alpha = \{gn \in VG^{i+1}_\beta \mid \beta \in A^i\}$, $i>0$, here, A^i is the subset of the names of the ith layer virtual groups. (This sentence means that the VG is formed from lower-layer groups.)

(2) If $gn_1 \in VG^i_\alpha \cap gn_1 \in VG^{i+1}_\beta$ and $gn_2 \in VG^i_\alpha \cap gn_2 \in VG^{i+1}_\beta$, then $gn_1 = gn_2$.

(3) Each VG has one and only one node (cvg) which takes coordinate functions. A gateway node is also a cvg in every layer VG to which the gateway node belongs except RVG. In RVG only one gateway node is also a cvg.

(4) Grid node p can join more than one VO.

(5) $PS=VO_1 \cup VO_2 \cup \ldots VO_{n1}$, Here, $n1$ is the number of virtual organization.

(6) If p satisfies the following condition: $p \in VO^m_{\alpha_m} \cap p \in VO^{m-1}_{\alpha_{m-1}} \cap \ldots \cap p \in VO^{m-k}_{\alpha_{m-k}}$, $m>=2$, the p is gateway node. It is expressed with symbol $gn(m, k, \alpha_{m-k}\ldots\alpha_{m-1}\alpha_m)$. The meanings of parameter values are as follows:

m is the layer order of VO in VGT ($gn \in$ VO). k is the number of layers in which the gateway node functions. $\alpha_{m-k}\ldots\alpha_{m-1}\alpha_m$ are the names of the virtual groups from $VO^m_{\alpha_m}$ to $VO^{m-k}_{\alpha_{m-k}}$.

Grid group management protocol (GGMP)

GGMP is a protocol used to manage membership of virtual group and virtual group tree. GGMP has two functions. Firstly, it manages membership of virtual groups and the dynamic virtual group tree. Secondly, when a gateway node fails or leaves, it selects a new one with the maximum weight value from all the on-line nodes in the group the gateway node is involved with. The details of the algorithm and the primitives and functions used in this paper are shown in Appendix.

To improve fault-tolerance, every member in a virtual group logically contains the group membership list, name, and cvg, and the membership list, name, and cvg of the groups immediately above and below. When there are any changes in the membership of a virtual group, such as a node joining or leaving, these changes are forwarded to the coordinator of the group, which forwards the information to all the members in the group, and to the groups in the neighboring levels. When one coordinator in a virtual group fails, another node in the same virtual group will replace it.

A node can use Query and Discovery Protocols, which are described in the following sub-sections, to find a suitable group to join. By using the Query and Discovery Protocols a VO can see the virtual group tree, and find the right place in the tree structure to add itself to. A VO can also create a new partial tree from any point of the VGT and add itself to the new tree path. For example, if in VGT there is a partial tree path such as ALL_Science:Biology (Biology is not VO), and a VO with the domain Fish wants to create the ALL_Science:Biology:Animal: Fish tree path, the VO first adds Animal partial tree path to ALL_Science:Biology, and then adds itself to the ALL_Science:Biology:Animal tree path. Then GGMP sets the gateway node of this VO as a member of the Biology and Animal groups.

As Appendix shows, the algorithm deals with the VO's or node's joining or leaving, and gateway node's leaving or failure.

In the case that a virtual organization as a whole joins VDHA Grid system, the algorithm first chooses a node with maximum weight as gateway node from this virtual organization, then it uses QDP protocol (defined in next section) to find the parts of interested in the structure of the virtual group tree, then sends the message of joining-requisition to the coordinator of the group the VO wants to attach to; after the coordinate approved, the virtual organization is added in the right place as a sub-tree. In the case that a virtual organization as a whole leaves VDHA Grid system, the coordinators of upper-layer virtual groups with which the virtual organization's gateway node is involved are replaced with the on-line nodes with maximum weight values among their groups. In the case that the gateway node leaves VDHA grid system, the gateway node is replaced with the on-line node with maximum weight value except of this leaving gateway node. In the case that a node joins or leaves a virtual organization, it forwards its requisition to a neighboring node. If the neighboring node accepts the node requisition, it forwards the joined/left node's information to gateway node, and the gateway node forwards this node information to all the nodes in the two neighboring layer groups. In the case that a gateway node fails to transmit messages between two neighboring layers exceeding a given time, the on-line nodes with maximum weight value (except of this failed gateway node) is chosen as a new gateway node.

In all of the above cases, the protocol updates the nodes' state data (node name, weight value, etc.) in the two neighboring layers.

The weight value is generally decided according to node's resources such as computational power and network bandwidth.

QUERY AND DISCOVERY PROTOCOLS

In VDHA, query and discovery protocols are used for querying and discovering some entities such as resources and services, virtual group name, node status, etc. Every node has resources and services which are described by WSDL or ontology languages, etc. Matching the request message is

done by the agent of the node which has the services. There are two kinds of QDP: Full search Query and Discovery Protocol (FSQDP), which searches all nodes to find nodes that match the request message, and Domain-Specific Query and Discovery Protocol (DSQDP), which searches nodes in only specific domains.

Full search query and discovery protocol (FSQDP)

FSQDP first finds the root virtual group, and then the coordinator of this group forwards the query message to all its members. All of these members forward in parallel the message down to the members of their low-layer groups until leaf virtual groups form as Fig.2 shows.

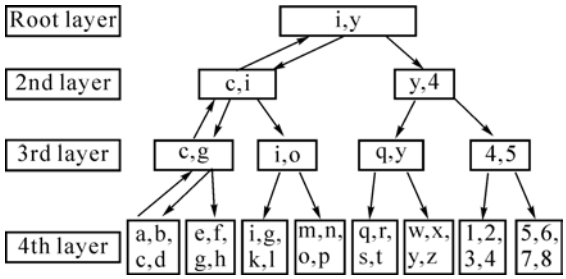

Fig.2 FSQDP searching process

Before we describe the algorithm of FSQDP, besides the primitives and functions in Appendix, we give the definition of message routing primitives and match function. The routing primitive cvg.Route_To_RVG forwards the message to the coordinator of the root virtual group. Its algorithm is as follows.

```
cvg.Route_To_RVG(qmessage)
1. If (Pnode(UPcvg(cvg)) ==∅
        Return cvg;
2. If (Pnode(UPcvg(cvg))<>Pnode(cvg))
      cvg.send(qmessage, UPcvg(cvg) );
3. UPcvg(cvg).Route_To_RVG(qmessage);
```

The routing primitive cvg.Route forwards the message in parallel to all nodes.

```
cvg.Route(qmessage)
1. cvg.send(qmessage, p_i∈VGroup(cvg) | p_i≠ cvg);
2. if (Type(VGroup(cvg)== VO)){
for (∀p_i∈VGroup(cvg)) {
     if (p_i.compare_service(qmessage) ==
      true){
        p_i.send(rmessage, ent),
        ent.send(rmessage, cli);}
  }
  }
else {
  for (∀p_i∈VGroup(cvg)) {
    LOWcvg(p_i, ayer(cvg)).Route(qmessage)
    }
  }
```

Function p_i.compare_service compares the requisition with the node p_i. If they are matched, it returns true, otherwise it returns false.

```
function p_i.compare_service(qmessage) {
  if p_i has the service indicated with qmessage
    return true;
  else
    return false;
}
```

As the Fig.2 shows, FSQDP first forwards the requisition message to the coordinator of the root virtual group, then forwards in parallel to all the nodes. If there are any matched nodes, the nodes send the result to the request client.

The algorithm of FSQDP is as follows:

```
/*suppose query message entity is expressed
with qmessage, result message entity expressed
with rmessage */
/*Suppose Client host (cli) uses entrance node
(ent), and ent is within a VO whose coordinator of
virtual group is cvg */
Step 1  cli.send(qmessage, ent)
Step 2  ent.send(qmessage, cvg)
Step 3  rcvg=cvg.Route_To_RVG(qmessage);
Step 4  rcvg.Route(qmessage);
```

Performance analysis of FSQDP

Let n^i_{vgmax} be the maximum number of nodes in

the ith layer virtual groups VG^i, that is,

$$n^i_{\text{vgmax}} = \max(N^i_{a_1}, \cdots N^i_{a_j} \cdots N^i_{a_n}),$$

$a_j \in$ name set of VG^i, $n = N^i_g$,

and suppose that the maximum time of sending a message is $T_{\max}$, so , due to forwarding message in parallel, maximum total time, space needs and message costs of searching all nodes with FSQDP is as given in Eqs.(1), (2) and (3).

$$T_{\max\,\text{all}} = L \times T_{\max} + T_{\max} \times \sum_i^L (n^i_{\text{vgmax}} - 1) \tag{1}$$

Here L is the number of layers. This formula is based on the following reason: FSQDP first forwards the message up to coordinator of root virtual group, so it has L hops. Then the coordinator of every virtual group forwards the message to the group members, so it takes (number of group members −1) forwarding (The coordinator does not need to forward the message to itself). Because the coordinators of the virtual groups in the same layer forward the message to the members of the virtual groups in parallel, the maximum time needed depends on the maximum number of virtual groups in this layer. Moreover, for the gateway node transfers down the message once a layer, the total time is as Eq.(1) shows.

$$S_{\max} = L \times n_{\text{vgmax}} \tag{2}$$

Here n_{vgmax} is the maximum number of nodes in a virtual group. The formula was deduced as follows: the node in a virtual group needs to keep the information of all members of the virtual group and its two neighbor layer groups. If the node is a gateway which is also a member of the root virtual group, after merging the same information it must keep information of all layer groups it is involved with.

The protocol first forwards the message to the coordinator of the root virtual group, which takes L hops; and then the coordinator forwards the message to the members of the group, and these members forward the message to members of their lower layer groups. Because the node does not forward the message to itself, we get the Eq.(3).

$$Message\_cost = L + N - 1 \tag{3}$$

Because the number of layers is small, so the time complexity, space complexity and message cost are given in Eqs.(4), (5) and (6)

$$T_{\text{complexity}} = O(n_{\text{vgmax}}) \tag{4}$$

$$S_{\text{complexity}} = O(n_{\text{vgmax}}) \tag{5}$$

$$Message\_cost_{\text{complexity}} = O(N) \tag{6}$$

If we suppose all virtual groups have the same number of nodes (n_{vg}) and the time for sending a message (T) is constant, then the number of layers is $\log_{n_{\text{vg}}} N$, or the number of a virtual group is $\sqrt[L]{N}$; from Eqs.(1) and (2) we have Eqs.(7) and (8).

$$T_{\max} = \log_{n_{\text{vg}}} N \times T \times n_{\text{vg}} \quad \text{or} \quad T_{\max} = L \times T \times \sqrt[L]{N} \tag{7}$$

$$S_{\max} = L \times n_{\text{vg}} \quad \text{or} \quad L \times \sqrt[L]{N} \tag{8}$$

Fig.3 shows the influence of number of nodes in a virtual group on the time response. If we suppose that the time for transferring a message is 0.1 second, the algorithm needs less than 100 seconds to search all 10 000 000 000 nodes with 200 nodes per virtual group and 5 layers ($\log_{200}10\,000\,000\,000$) layers.

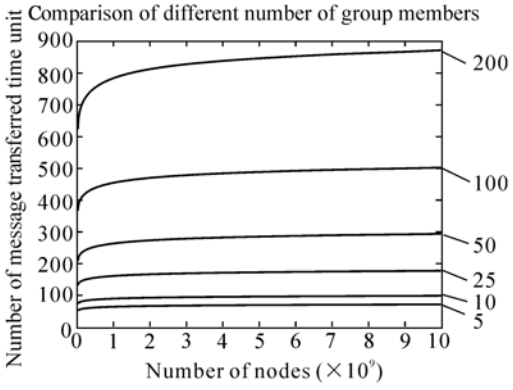

Fig.3 Influence of number of nodes in a virtual group on the time response

Fig.4 shows the influence of number of layers in a virtual group on the time response. Except for 3 layers, the response time varies little with the increase of nodes. Because more layers will increase the cost of Grid Group Management, the 4−6 number of layers is the best.

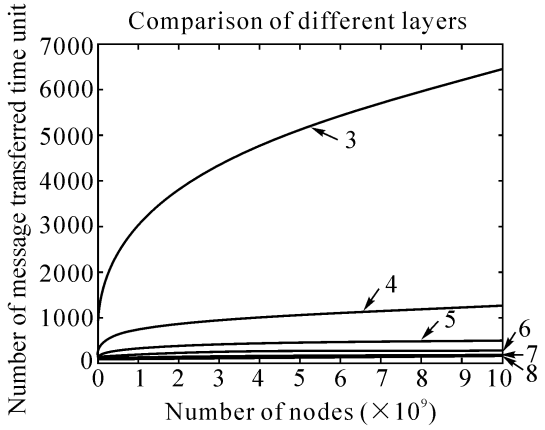

Fig.4 Influence of number of layers on the time response

How about the traffic? Does the algorithm cause a network jam?

This problem is still under study. If we suppose that a message length is L_p, and traffic load is distributed averagely in N_l links, then we have maximum traffic per link as shown in Eq.(9).

$$Traffic_{link} = (L_p \times (N + L - 1)) / (L \times T \times \sqrt[L]{N} \times N_l) \quad (9)$$

From Eq.(9), we can approximately calculate the use of bandwidth. For example, if L_p is 1000 bits, L is 5, N is 1 000 000, T is 0.1 second, and N_l is 1000, then maximum traffic per link is 126 kb/s.

Domain-Specific Query and Discovery Protocol (DSQDP)

FSQDP is effective, but may be bandwidth-consuming. Domain-Specific Query and Discovery Protocol does not have this problem. To use this protocol, the object of the virtual group must maintain a catalogue with classifying services from general to detail. This may be done by the nodes' joining the proper virtual group of Grid system. The protocol only searches the nodes whose catalogue matches the request group keywords as Fig.5 shows.

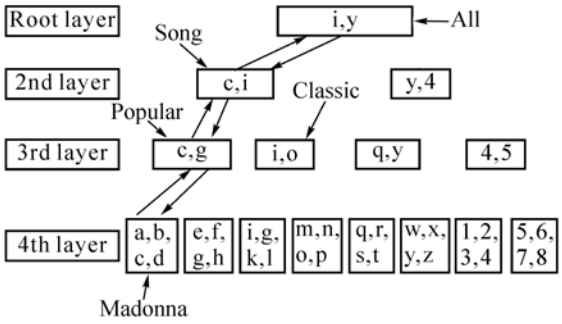

Fig.5 DSQDP searching process

The algorithm is similar to FSQDP. The only difference is that DSQDP only searches the matched node which has the same keyword with the requisition message. The difference is described as follows:

```
cvg.Route(qmessage)
1. cvg.send(qmessage, p_i∈VGroup(cvg) | p_i≠ cvg);
2. if (Type(VGoup(cvg)== VO)){
   for (∀p_i∈VGroup(cvg)) {
   if (p_i.compare_service (qmessage) == true){
      p_i.send(rmessage, ent),
      ent.send(rmessage, cli);}
   }
}
   else {
      for (∀p_i∈VGroup(cvg)) {
      if(keyword(qmessage, Layer(cvg)) ==
       keyword(GroupID(cvg)))
LOWcvg(p_i, Layer(cvg)).Route(qmessage)
          }
        }
```

In the above pseudo-code, function keyword (qmessage, Layer (*cvg*)) returns keyword of cvg in the layer of Layer (*cvg*), function keyword (Group ID(*cvg*)) returns group's keyword.

Performance analysis of DSQDP

The terms are the same as those in the above section. We have:

$$T_{\max} = 2L \times T_{\max} + T_{\max} \times (n_{\mathrm{vg}\max} - 1) \quad (10)$$

This is because only nodes in the virtual organization, which match the keyword with the requisition message, are needed to search. And it first forwards the message up to the coordinator of root virtual group, then forwards down the message to the virtual organization.

For the same reason as Eq.(2), we have

$$S_{\max} = L \times n_{\mathrm{vg}\max} \quad (11)$$

For the same reason as Eq.(10), we have

$$Message\_cost_{\max} = 2L + n_{\mathrm{vg}\max} - 1 \quad (12)$$

Obviously, we have the time complexity, space complexity and message cost as follows:

$$T_{\mathrm{complexity}} = O(n_{\mathrm{vg}\max}) \quad (13)$$

$$S_{\mathrm{complexity}} = O(n_{\mathrm{vg}\max}) \quad (14)$$

$$Message\_cost_{\mathrm{complexity}} = O(n_{\mathrm{vg}\max}) \quad (15)$$

This protocol is effective and message cost is low. But, the resources and services must cluster according to hierarchical keywords.

CONCLUSION

The services discovery based on VDHA we presented is scalable, autonomous, efficient, reliable and rapid in response, and searching fully. FSQDP discovers the nodes that match the request message from all N nodes, which has time complexity $O(\log N)$, space complexity $O(n_{\mathrm{vg}})$ (n_{vg} is number of nodes of each virtual group), and message-cost $O(N)$. Whereas, when the services are clustered as classification, we can use DSQDP to discover services with time complexity $O(n_{\mathrm{vg}})$, space complexity $O(n_{\mathrm{vg}})$, and message-cost $O(n_{\mathrm{vg}})$. With each virtual group having 200 nodes and a 4 to 6 layer virtual group tree, the protocols are suitable for more than 10 billions of nodes.

The services discovery based on VDHA is fully decentralized, without the disadvantage of single point of failure. There is no need for nodes to know all global names of groups or node identification, etc., because the groups are organized as a virtual group tree and the group and node properties can be obtained by Query and Discovery Protocol.

Service discovery based on VDHA is un-related to services description languages, because it uses local agents of nodes to match the services. The only thing for agents to do is obeying the specification of service discovery message format and match services according to the local service description.

References

Chander A., Dawson, S., Lincoln, P., Stringer-Calvert, D., 2002. NEVRLATE: Scalable Resource Discovery. Proceedings CCGRID. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., 2001. Web Services Description Language (WSDL) 1.1. W3C, Note 15, http://www.w3.org/TR/wsdl.

Foster, I., Kesselman, C., 1997. Globus: a metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, **11**(2):115-128.

Foster, I., Kesselman, C., Tuecke, S., 2001. The anatomy of the grid: enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, **15**(3):200-222.

Foster, I., Kesselman, C., Nick, M.J., Tuecke, S., 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/ research/papers/ogsa.pdf.

Grid Web Services Workshop, 2001. http://gridport.npaci.edu/workshop/webserv01/agenda.html.

Huang, L.C., Zhou, X., Wu, Z.H., Pan, Y.H., 2002a. Virtual and Dynamic Hierarchical Architecture and Its Usage in An E-Science Application for Providing Knowledge Services. *In*: Proceedings of the Joint International Computer Conference 2002(JICC2002). Zhejiang University Press, Hangzhou, p.159-165.

Huang, L.C., Wu, Z.H., Pan, Y.H., 2002b. Virtual and Dynamic Hierarchical Architecture for E-Science and Related Protocols. *In*: Proceedings of 2002 International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES2002).

Huang, L.C., Wu, Z.H., Pan, Y.H., 2002c. Knowledge Services Provider Model Based on Virtual and Dynamic Hierarchical Architecture. *In*: Proceedings of

2002 International workshop on Grid and Cooperative Computing(GCC2002). Publishing House of Electronics Industry, Beijing, p.297-311.

Iamnitchi, A., Foster, I., 2001. On Fully Decentralized Resource Discovery in Grid Environments. International Workshop on Grid Computing .

Rana, O.F., Walker, D.W., 2002. Service Design Patterns for Computational Grids. *In*: Patterns and Skeletons for Parallel and Distributed Computing, edited by Fethi A. Rabhi and Sergei Gorlatch, Springer-Verlag.

Ratnasamy, S., Francis, P., Handley, K., Karp, R., Shenker, S., 2001. A Scalable Content-Addressable Network. *In*: Proceedings of ACM SIGCOMM 2001.

Roure, D.D., Jennings, N., Shadbolt, N., 2001. Research Agenda for the Semantic Grid: A Future E-Science Infrastructure. http://www.semanti.grid.org/v1.9/semgrid.pdf.

Rowstron, A., Druschel, P., 2001. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Pear Systems. *In*: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware).

Stoica, I., Morris, R., Karger, D., Kaashoek, F.M., Balakrishnan, H., 2001. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *In*: Proceedings of ACM SIGCOMM2001.

UDDI: Universal Description, Discovery and Integration, 2001. http://www.uddi.org/.

Wahl, M., Howes, T., Kille, S., 1997. Lightweight Directory Access Protocol (v3), IETF RFC 2251.

Web Services Flow Language (WSFL) Version 1.0, 2001. http://www4.ibm.com/software/solutions/Web-sevices/pdf/WSFL.pdf.

Web Services for Business Process Design, 2001. http://www.gotdonet.com/team/xml_wsspecs/xlang-c/default.htm.

APPENDIX

Primitives, functions and GGMP algorithm

Table 1 Primitives and functions

Description	Meaning
Primitive sender.send (mesage, receiver)	Sender sends message to receiver
Primitive sender.send (message, receiver∈Set)	Sender sends message to all the receiver belong to Set
Primitive remove (p, VG)	p is removed from VG, and all $p \in VG$ know that p is removed from VG
Primitive choose_new_cvg (p, VG, condition)	p is chosen from virtual group VG according to the condition
Primitive add (p, VG)	p is added into VG, and all $p \in VG$ know that p is added into VG
Function Pnode(cvg)	It returns node ID of cvg
Function GroupID(VGroup(p))	It returns this group's identification
Function Layer ($var^m_{\alpha_m}$)	It returns the layer of cvg, or gn
Function UPcvg(cvg)	It returns up-layer cvg
Function LOWcvg ($cvg^m_{\alpha_m}$)	It returns low-layer cvg
Function LOWcvg (p, Layer(cvg))	It returns cvg of the next low layer
Function Vgroup ($cvg^m_{\alpha_m}$)	It returns the virtual group $cvg^m_{\alpha_m}$ is within
Function Type (VG)	It returns the type of virtual group
Function BOTTOM_gn(gn)	It returns gateway node in the bottom layer
Function TOP_gn(gn)	It returns gateway node in the top layer
Function Down_gn($gn^i_{\alpha_i}$)	It returns gateway node in the next lower layer
Function UP_gn($gn^i_{\alpha_i}$)	It returns gateway node in the next upper layer
Primitive gn.Reselect_Gateway Node_Coordinator ()	It selects a new coordinator
Primitive gn.Down_Update ()	It updates the changed information downwards bottom layer
primitive gn.Up_Update ()	It updates the changed information upwards top layer

The pseudo-codes of several primitives and functions are listed as follows:

Function UPcvg($cvg^m_{\alpha_m}$) {

if ($\exists$($cvg^{m-1}_{\alpha_{m-1}} \in VG^{m-1}_{\alpha_{m-1}} \cap$ Pnode($cvg^m_{\alpha_m}$) $\in VG^{m-1}_{\alpha_{m-1}}$))

return $cvg^{m-1}_{\alpha_{m-1}}$;

else

return $\varnothing$; }

Function LOWcvg($cvg^m_{\alpha_m}$){

if ($\exists$($cvg^{m+1}_{\alpha_{m+1}} \in VG^{m+1}_{\alpha_{m+1}} \cap$ Pnode($cvg^m_{\alpha_m}$) $\in VG^{m+1}_{\alpha_{m+1}}$))

return $cvg^{m+1}_{\alpha_{m+1}}$;

else

return $\varnothing$; }

Function LOWcvg(p, Layer(cvg)) {
if ($\exists$(Layer (cvg_1)==Layer(cvg) +1$\cap p$== Pnode(cvg_1)))
return cvg_1;
else
return $\varnothing$; }

Function VGroup ($cvg^m_{\alpha_m}$) {
if ($cvg^m_{\alpha_m} \in VG^m_{\alpha_m}$)
return $VG^m_{\alpha_m}$; }

Function Type(VG) {
if VG is a VO
return VO; // virtual organization type
else
return VG; // pure virtual group }

Function BOTTOM_gn(gn){ /*suppose $gn(m, k, \alpha_{m-k}\ldots\alpha_{m-1}\alpha_m)$ */
return $gn^m_{\alpha_m} \in VG^m_{\alpha_m}$;}

Function TOP_gn(gn){ /*suppose $gn(m, k, \alpha_{m-k}\ldots\alpha_{m-1}\alpha_m)$ */
return $gn^{m-k}_{\alpha_{m-k}} \in VG^{m-k}_{\alpha_{m-k}}$; }

Function Down_gn($gn^i_{\alpha_i}$) {/*suppose $gn(m, k, \alpha_{m-k}\ldots\alpha_{m-1}\alpha_m)$ */
if (i>m−1)
return $\varnothing$;
else
return $gn^{i+1}_{\alpha_{i+1}} \in VG^{i+1}_{\alpha_{i+1}} \cap$ Pnode($gn^i_{\alpha_i}$) == Pnode($gn^{i+1}_{\alpha_{i+1}}$);}

Function UP_gn($gn^i_{\alpha_i}$) {/*suppose $gn(m, k, \alpha_{m-k}\ldots\alpha_{m-1}\alpha_m)$ */
if (i<m−k+1)
return $\varnothing$;
else
return $gn^{i-1}_{\alpha_{i-1}} \in VG^{i-1}_{\alpha_{i-1}} \cap$ Pnode($gn^i_{\alpha_i}$) == Pnode($gn^{i-1}_{\alpha_{i-1}}$);}

Primitive gn.Reselect_GatewayNode_Coordinator () {
if $gn \neq \varnothing$ {
if (Pnode(gn) ==$cvg\in$VGroup (gn))
choose_new_cvg(cvg, VGroup(gn), ($cvg\in$VGroup (gn) $\cap$ Pnode(cvg) $\neq$ Pnode(gn) $\cap$ cvg_w =Maxium of $p_{iw}, p_i \in$VGroup(gn) and online));
remove(gn, VGroup (gn));
if (Layer(cvg)<>Layer(TOP_gn(gn)))
add(cvg, VGroup(UP_gn(gn)));
Down_gn(gn). Reselect_GatewayNode_Coordinator();
}}

Primitive gn.Down_Update (){
if ($gn \neq \varnothing$) {
cvg.send(state_table of VGroup(cvg)_message, LOWcvg(cvg));
LOWcvg(cvg).send(state_table of VGroup(cvg)_message, $p\in$VGroup(LOWcvg(cvg)));
Down_gn(gn).Down_Update(); } }

Primitive gn.Up_Update(){
if ($gn \neq \varnothing$) {
cvg.send(state_table of VGroup(cvg)_message, UPcvg(cvg));
UPcvg(cvg).send (state_table of VGroup(cvg)_message, $p\in$VGroup(UPcvg(cvg)));
UP_gn(gn).Up_Update();}}

GGMP algorithm

/*p_w means that Grid node has weight value w. Weight values of nodes are assigned as several classes according to nodes' resources etc.*/

/*Suppose the gateway node is $gn(m, k, \alpha_{m-k}\ldots\alpha_{m-1}\alpha_m)$*/

/*Suppose a gn^i_α For all $p_i\in$VGroup(gn^i_α), pi contains three tables (If p_i is in the root group or leaf group (VO), then two tables).

state table: including Group member list: node names (IP Addresses) of $\forall p_i\in$VGroup(gn^i_α), and group_name (VGroup(gn^i_α)), $cvg\in$VGroup(gn^i_α), etc .

Up state table: including up layer Group member list: node names (IP Addresses) of $\forall p_i\in$VGroup(UP_gn(gn^i_α)), and group_name(VGroup(UP_gn(gn^i_α))), $cvg\in$Vgroup(UP_gn (gn^i_α)), etc.

Down state table: down layer Group member list: node names (IP Addresses) of $\forall p_i\in$VGroup(Down_gn (gn^i_α)) , and group_name(VGroup(Down_gn (gn^i_α))), $cvg\in$Vgroup (Down_gn(gn^i_α)), etc .

These tables are needed to be synchronized and consistent, that is, every node keeps a copy of the status*/

while(true) {
switch(event) {
case: a VO_α joins VDHA Grid system :

/* a VO as a whole to join VDHA Grid , Root virtual group's symbol is VG^1_{top} */

choose_new_cvg (gn, VO_α, ($gn \in VO_\alpha \cap gn_w$=Maxium of $p_{iw}, p_i \in VO_\alpha$ and online));

/* If p_{iw}, p_{jw}, etc. are with the same value, random node is chosen. */

set $cvg = gn$; /* cvg is the coordinator of VO_α that is, $cvg \in VO_\alpha$ */

cvg uses QDP protocol to find the interested parts of the structure of virtual group tree such as VG^k_β ; /*Choose VG^k_β as an upper-layer virtual group to join*/

cvg.send(JOIN_MESSAGE, cvg^k_β);

if (cvg^k_β accepts the requisition) add (Pnode(cvg), VG^k_β) /* cvg can form a sub tree of virtual groups, and can join VG^k_β as a whole */

cvg.send(state_table of VGroup(cvg)_message, Upcvg (cvg));

UPcvg(cvg).send (state_table of VGroup(cvg)_message, $p \in$VGroup(UPcvg(cvg))); /*update down state table */

UPcvg(cvg).send (state_table of VGroup(UPcvg(cvg))_message, cvg);

cvg.send(state_table of VGroup(UPcvg(cvg))_message, $p \in$(VGroup(cvg)); /*update up state table */

case: a VO_α leaves from VDHA Grid system:
gn=Pnode(TOP_gn($gn \in VO_\alpha$);
gn.Reselect_GatewayNode_Coordinator ();
Delete VO_α; // delete VO_α
gn=Pnode(BOTTOM_gn($gn \in VO_\alpha$);
gn.Up_Update ();
gn=Pnode(TOP_gn($gn \in VO_\alpha$));
gn.Down_Update ();

case: gn leaves VDHA Grid system:
VG=VGroup(BOTTOM_gn(gn))
gn. Reselect_GatewayNode_Coordinator ();
set new $gn_1 = cvg \in VG$;
gn=Pnode(BOTTOM_gn($gn \in VG$));
gn.Up_Update ();
gn=Pnode(TOP_gn($gn \in VG$);
gn.Down_Update ();

case: cvg fails to receive messages from $p \in$VGroup(cvg), $p \in$VGroup(UPcvg(cvg)) and $p \in$VGroup(Lowcvg(cvg)) exceeding a given time
set gn=Pnode(cvg);
VG=VGroup(BOTTOM_gn(gn))
gn.Reselect_GatewayNode_Coordinator ();
set new $gn_1 = cvg \in VG$;
add (Pnode(gn),VG); /*change previous gn to an ordinary node.*/
gn = Pnode(BOTTOM_gn($gn \in VO_\alpha$));
gn.Up_Update ();
gn=Pnode(TOP_gn($gn \in VO_\alpha$));
gn.Down_Update ();

case: a node p joins a VO
p.send(REQUISITION_MESSAGE, $p_{neighbor}$);
/*$p_{neighbor}$ is the neighboring node in the VO, which is known to p */
$p_{neighbor}$.send(REQUISITION_MESSAGE, $cvg \in VO$);
if (cvg accepts the requisition) add(p, VO);
cvg.send(p_joins_message, $p \in$VGroup(cvg)); /* update state table */
cvg.send(copy_all_table_message, p); /* update state table */
cvg.send(p_joins_message, UPcvg(cvg));
UPcvg(cvg).send (p_joins_message, $p \in$VGroup(UPcvg(cvg))); /*update up state table */

case: a node p leaves from a VO:
p.send (REQUISITION_MESSAGE, $cvg \in VO$);
if (cvg accepts the requisition) remove(p, VO);
cvg.send(p_leaves_message, $p \in$VGroup(cvg));
/* update state table */
cvg.send(p_leaves_message, UPcvg(cvg));
UPcvg(cvg).send (p_leaves_message, $p \in$VGroup(UPcvg(cvg))); /*update up state table */
}
}